False Negative Problem of Counting Bloom Filter

Deke Guo, *Member, IEEE,* Yunhao Liu, *Senior Member, IEEE,* Xiangyang Li, *Senior Member, IEEE,* and Panlong Yang, *Member, IEEE,*

Abstract—Bloom filter is effective, space-efficient data structure for concisely representing a data set and supporting approximate membership queries. Traditionally, researchers often believe that it is possible that a Bloom filter returns a false positive, but it will never return a false negative under well-behaved operations. By investigating the mainstream variants, however, we observe that a Bloom filter does return false negatives in many scenarios. In this work, we show that the undetectable incorrect deletion of false positive items and detectable incorrect deletion of multi-address items are two general causes of false negative in a Bloom filter. We then measure the potential and exposed false negatives theoretically and practically. Inspired by the fact that the potential false negatives are usually not fully exposed, we propose a novel Bloom filter scheme which increases the ratio of bits set to a value larger than one without decreasing the ratio of bits set to zero. Mathematic analysis and comprehensive experiments show that this design can reduce the number of exposed false negatives as well as decrease the likelihood of false positives. To the best of our knowledge, this is the first work dealing with both the false positive and false negative problems of Bloom filter systematically when supporting standard usages of item insertion, query, and deletion operations.

Index Terms—Bloom filter, False negative, multi-choice counting Bloom filter.

1 Introduction

A Bloom filter (BF) [1] is a space-efficient data structure for representing a set and supporting membership queries. It outperforms other efficient data structures such as binary search trees and tries, as the time needed to add an item or check whether an item belongs to the set is constant irrespective to the cardinality of the set. For these advantages, BF has been extensively used in database as well as networking applications [2], [3], web cache sharing [4] and routing on overlay networks [5], [6], [7]. Moreover, BF has great potential to summarize streaming data in the main memory [8], store the states of a large number of flows in the on-chip memory of the routers [9], and speed up the statistical-based Bayesian filters [10]. To make it more effective and efficient, BF has been improved from different aspects for a variety of applications. Some important variations include the compressed Bloom filter [11], counting Bloom filter (CBF) [4], distance-sensitive Bloom filter [12], space-code Bloom filter [13], spectral Bloom filter [14], generalized Bloom filter [15], and Bloomier filter [16].

Despite the aforementioned benefits offered by BF, a BF may yield a *false positive* due to hash collisions, for which it wrongly determines that an item belongs to a data set when the item is actually not. The cause is that all bits related to the item were previously set to 1 by other items in the data set. A possible way to deal with hash collisions is to design perfect hash functions. This is only possible for a static data set without item insertion and deletion after deployment. In reality, however, BF and its variants are widely used to represent both static and dynamic data sets. That said, the data set is often unknown in advance, therefore, it is impossible to design perfect hash functions. Thus, the false positive is unavoidable in a BF and its variants, and hence many efforts were made to reduce the probability of false positive during the past years [17], [18], [19], [20].

For a static data set X, it is not allowed to perform data insertion or deletion operations after we represent it as a BF. Thus the bit vector of a filter always reflects the data set correctly. The membership queries based on BF never produce a false negative in this scenario. By handling a dynamic data set, a deletion operation might hash an item to be deleted and resets the related bits to 0. It may set a location to 0 to which is also hashed by other items in the set X. In such a case, the filter no longer reflects the data set correctly, thus producing a false negative. To address the problem, Fan et al. propose the CBF [4], in which each entry is not a single bit but rather a small counter consisted of several bits. When an item is added (or deleted), the corresponding counters are incremented (or decremented respectively). By assuming that *false negatives* rarely happen for a CBF, researchers thus pay less attention to false negative problem of CBF. In this work, we reveal that a CBF indeed can result in false negative items in many networking applications. In many cases, the false negative is more serious than the false positive, because it wrongly decides that an item does not belong to a data set when it does, and rejects desired objects, leading to intolerable consequences [8], [21], [22].

To the best of our knowledge, we are the first to explore the root cause of the false negative problem of CBF in standard

• D. Guo is with the Key laboratory of C^{4}ISR Technology, National University of Defense Technology, Changsha, China, 410073. E-mail: guodeke@ieee.org.
• Y. Liu is with the Computer Science Department, Hong Kong University of Science and Technology, Hong Kong, China. E-mail: liu@cse.ust.hk.
• X. Li is with Institute of Computer Application Technology, Hangzhou Dianzi University, Hangzhou, PRC, and Department of Computer Science, Illinois Institute of Technology, Chicago, USA. E-mail: xli@cs.iit.edu.
• P. Yang is with the Institute of Communication Engineering, P.L.A University of Science and Technology, Nanjing Jiangsu, China. E-mail:plyang@computer.org.

usage of item insertion, query and deletion operations. We also measure the potential and exposed false negative items from the aspects of theory and practice. We propose two principles to minimize the number of exposed false negative items. We also design a variant of CBF to reduce the number of exposed false negative items. The main contributions of this paper are as follows:

1) We show that a false positive can trigger a deletion of a false positive item, and result in at least one multi-address item. Both of the two cases cause an incorrect item deletion operation, and lead to potential false negative items.
2) We reveal that the resulting false negative items are usually not fully exposed in consequent queries. We also measure the potential and exposed false negative items caused by an incorrect item deletion operation.
3) We propose two fundamental principles to make potential false negatives unexposed whenever possible. Our design is able to increase the ratio of bits set to a value larger than one in a BF without decreasing the ratio of bits set to zero.
4) We propose an enhanced BF scheme which can reduce about $50-80\%$ of exposed false negative items in BF. Through comprehensive experiments and mathematic analysis, we show that our design achieves desired properties.

The rest of this paper is organized as follows. Section 2 briefly introduces the BF and related work, and discusses the root cause of false negative items in a CBF. Section 3 measures the potential and exposed false negatives. Section 4 presents a variant of CBF. We discuss our experimental methodology and evaluate this design in Section 5, and conclude the work in Section 6.

2 PRELIMINARIES

We first review some related concepts of Bloom filers and then discuss why false negative items can happen in counting Bloom filers. Some related work are also briefly reviewed.

2.1 Bloom filter and counting Bloom filter

A set X of n items is represented by a BF using a vector of m bits which are initially set to 0. A BF uses k *independent* random hash functions $h_1, h_2, \cdots, h_k$ with a range $\{1, ..., m\}$. When inserting an item x to X, all bits of a Bloom filter address $\mathrm{Bfaddress}(x)$ (consisted of k addresses $h_i(x)$ for $1 \leq i \leq k$) will be set to 1. To answer a membership query for any item x, users check whether all bits $h_i(x)$ are set to 1. If not, x is not a member of X. If yes, we assume that x is a member of X, although we might be wrong in some cases. Hence, a BF may yield a *false positive* due to hash collisions, in which all bits of $\mathrm{Bfaddress}(x)$ were set to 1 by other items in set X [1]. In BF, no item deletions are allowed. CBF [4] provides a way to implement a delete operation on a BF without regenerating the filter afresh. In a CBF each item of its bit vector is extended from being a single bit to being a L-bits counter, and L=4 usually suffices[1]. The operation of item insertion is extended to increment the value of each respective counter (defined by $h_i(x)$) by one. The operation of item deletion decrements the value of each respective counter by one.

Once a data set X is represented as a BF, user can determine whether an item x belongs to X by querying the filter instead of set X. A membership query based on BF produces one of the following results.

1) The judgment always matches the fact. In other words, if $x \in X$, then $\forall i \in \{1, 2, ..., k\}$ we have $h_i(x) \neq 0$; and if $x \notin X$ then $\exists i \in \{1, 2, ..., k\}$ such that $h_i(x) = 0$.
2) Although x does not belong to X, the judgment returns a reversed result, called a *false positive* judgment. In other words, if $x \notin X$ then $\forall i \in \{1, 2, ..., k\}$ satisfying that $h_i(x) \neq 0$. Here, the item x is called a *false positive item*.
3) Although x belongs to X, the judgment returns a reversed result, called a *false negative* judgment. In other words, if $x \in X$ then $\exists i \in \{1, 2, ..., k\}$ satisfying that $h_i(x) = 0$. Here, the item x is called a *false negative item*.

Let n be the number of items in the set X, and p denote the probability that a random bit of the corresponding BF is 0. By assuming that all hash functions produce values uniform randomly from $[1, m]$, clearly $p = (1 - 1/m)^{kn} \approx e^{-n \times k/m}$, as $n \times k$ bits are randomly selected, with probability $1/m$ in the process of adding each item. The probability that a random bit is 1 is therefore $1 - (1 - 1/m)^{kn}$. Now we test membership of an item x_1 that is not in the X. Each of the k bits of the $\mathrm{Bfaddress}(x_1)$ is 1 with a probability as above. The probability of all of the k bits being 1, which would cause a false positive, is then

$$f(m, k, n) = (1 - p)^k \approx (1 - e^{-k \times n/m})^k. \qquad (1)$$

These results about the membership query based on BF also hold for the membership query based on CBF. Recall that the basic component of a CBF is a counter instead of a bit. Unless explicitly stated, we use the notations defined for BF to explain the same concept in CBF in the rest of this paper.

2.2 Related work

It is well known that false negative items do not arise at all in a BF if the BF always correctly reflects the membership information of a data set represented by it. Unfortunately, this essential condition is often destroyed by many non-standard behaviors.

A BF is replicated by multiple nodes to support efficient protocols in distributed systems. The replicas might become stale because the changes of a BF cannot be spread quickly to all replicated BFs. Hence, false negative items are produced. The false negative and false positive in the stale replicas of a BF are analyzed in [22].

BF is widely used to represent stream data since the allocated space is rather small compared to the size of the stream. When a large number of items arrive, the false positive

1. When k is chosen as optimal $\ln 2 \cdot \frac{m}{n}$, $L = 4$ suffices since the average load of a counter is $\ln 2$ and the probability that a counter has load $15 = 2^4 - 1$ is around 6.8×10^{-17}.

rate increases to an unreasonable value quickly. To address this issue, the stable BF [8] attempts to drop the older data by randomly evicting some information from it even they do not know which part is stale.

Based on the following two assumptions, the retouched BF [21] removes entire false positives or partial serious false positives by resetting individually chosen bits to 0. First, all false positives and those serious ones can be identified after the BF has been constructed. Second, the application can tolerate false negative items. The retouched BF decreases or avoids false positives, but actively produces many false negative items. Note that it is uncommon for applications to satisfy these two assumptions, especially the second one.

In summary, the stable BF and retouched BF adopt specific bit cleaning operations to deal with application-specific problems, yet introduce many false negative items. These are essentially non-standard usage of BF. If those applications use CBF instead of BF, the non-standard behaviors will produce similar results. In this work, we find that a CBF might produce false negative items even in standard usages of item insertion, query, and deletion operations. Specifically, we discuss false negative items caused by an incorrect item deletion operation triggered by once false positive in a CBF.

2.3 False negative items in counting Bloom filters

It is well-known that CBF supports item deletion operation at the cost of consuming more spaces than BF. Many researchers have studied the scenario where item deletions occur and are always correct. A common precondition is that a data set is stored together with a corresponding CBF to ensure that only the deletion instructions of items in the data set are set to the CBF. That is, an item can be deleted from a CBF only if it has been inserted into it. This precondition, however, generally is not satisfied because it deviates from the objective to replace a data set with a CBF, especially in some network applications as follows. These applications just maintain a CBF without keeping the data set.

As mentioned in [9], routers and networking devices are likely to evolve to be more application-aware. Many existing routers and switches begin to monitor traffic flows by keeping state about TCP connections for security violations and to steer traffic based on packet content. Specifically, the intrusion detection devices and packaged firewalls keep state for each TCP connection in order to detect security violations. The application level QoS devices track the state of each flow to provide more discriminating QoS to applications by steering traffic based on packet content, such as video congestion control [23] and identifying Peer-to-Peer traffic [24]. For high-speed networking devices, the *challenge* to track state for each flow on-chip without resorting to slow off-chip memories is the limited on-chip memory. For example, consider a router keeping track of 1 million (a number found in many studies [25]) TCP connections. If 100 bits are used to track each connection, it costs 100 Mbits memory, which is impractical using on-chip memory.

To deal with such challenging issues in many networking applications, Bonomi et. al use a CBF to track the state of the same number of concurrent flows such that the needed on-chip memory can be reduced by a factor of 5 to 20 Mbits in [9]. Here, it is unnecessary and impractical for a networking device to store states of flows together with a corresponding CBF. In such scenarios, the CBF cannot ensure that only an item, which has been inserted into it, can be deleted from it. That is, the CBF also decrements the value of each respective counter by one when it receives a deletion instruction for a *false positive item*, and thus produces possible *false negative items*.

Throughout this paper, we call this type of item deletion as *incorrect deletion of a false positive item*. In reality, several intentional or unintentional behaviors might trigger an incorrect item deletion. For example, adversaries can issue an instruction to delete an item x after detecting that x is a false positive item, and intentionally produce potential false negative items. For another example, the id of a flow is inserted into a CBF when its first packet arrives, subsequent packets check whether the flow has been recorded, and the flow is deleted when the last packet is processed. In some cases, a network device might receive a subset of packets without the first packet of a flow. If the id of this flow is a false positive item, the last packet can result in false negative items unintentionally.

After discussing the incorrect deletion of a false positive item, we find that an item deletion in a CBF might be incorrect due to the multi-address problem even when the item has been inserted into the CBF. This kind of item deletion might cause potential false negative items, and is referred as the *incorrect deletion of a multi-address item*. In reality, we are aware of at least the following representative scenarios about this kind of incorrect deletion.

First, a CBF may respond multiple $\mathrm{Bfaddress}(x)$ for a query with an item $x \in X$ as input. For example, Bonomi et al. use a CBF to store and track the states of many flows associated with unique $flow\text{-}id$ at network devices [9]. They append the state value of a flow to the $flow\text{-}id$ as an item, and then add it in a CBF. When the state of a flow is retrieved or updated, one must perform a membership query for each combination of the $flow\text{-}id$ and possible state value. In such a situation, a flow may appear to have multiple states because of one or more false positives in the CBF. It is difficult for the CBF to determine which is the right one. Thus, a state update operation may cause a wrong item deletion operation, and then results in false negative items. In reality, each flow transmits its state frequently during its life cycle, and the number of flows tracked by a network device could be huge. Thus, the number of cumulative false negative items is no trivial, and their impact cannot be omitted.

Second, it seems that multiple CBFs represent a same item even if only one CBF does so in several variants of CBF, such as the dynamic CBF and scalable CBF [17], [20]. A dynamic CBF uses a CBF to represent a dynamic data set. If the cardinality of the dynamic data set reaches a predefined threshold, it will allocate another CBF to represent the following data items, and so on. An item x belonging to the dynamic data set may appear in multiple CBFs due to the false positives, and a CBF might wrongly delete the item x, leading to false negative items. A scalable CBF adopts a similar concept and suffers the same problem.

In summary, the incorrect deletion of a false positive item, or a multi-address item are the root causes of false negative items in a CBF and its variants. They affect the CBF and its variants in the same way by decrementing the value of respective counters, and are equivalent in nature. The only difference is that the incorrect deletion of a false positive item is *undetectable*, while the incorrect deletion of a multi-address item is *detectable* in advance. In this work, we only focus on the false negative problem in the CBF, and leave the study of the same issue in variants of the CBF as a future work.

3 Measurement of false negative items

In this work, we only discuss the incorrect deletion of a false positive item since it is equivalent to the incorrect deletion of a multi-address item. Specifically, we first measure the expected value of the number of false negative items caused by an incorrect deletion of a false positive item. We observe that the potential false negative items may be not exposed to the upcoming membership queries immediately. Inspired by the observation, we also measure how many false negative items will be exposed in theory and practice. We finally propose two principles to reduce the number of exposed false negative items, even make all potential false negative items become unexposed.

Before measuring the false negative items, let us consider the four rules to delete an item x from a respective CBF of a set X.

1) If a membership query for an item $x \in X$ responses a right judgment, the CBF performs the item deletion operation by decrementing respective counters by one.
2) If a membership query for an item $x \in X$ responses a false negative, the CBF rejects the item deletion operation. It shows that the CBF does not reflect the set X correctly.
3) If a membership query for $x \notin X$ responses a right judgment, the CBF omits the item deletion operation.
4) If a membership query for $x \notin X$ responses a false positive judgment, the CBF still performs the item deletion operation. Consequently, the CBF does not represent the set X correctly after the operation.

The event mentioned in the fourth rule is the root cause of subsequent false negative judgments, and furthermore it can bring in the event illustrated in the second rule. According to Formula 1, the false positive probability of a CBF should decrease in theory if it performs an item deletion operation. Thus, the second rule may increase the false positive probability of a CBF as an item is not deleted, although it should be deleted. In summary, the fourth rule not only produces false negative items directly but also may increase the probability of false positive judgments indirectly.

3.1 Potential false negative items in theory

Given a set X and its CBF, an incorrect deletion of a false positive item causes the affected counters decrease at least by one. The resulting false negative items will be found through the following steps. First, we delete each item, whose CBF address overlaps with those affected counters and all counters of its CBF address are larger than 0, from the CBF and X correctly. Second, we perform a round of set membership queries for each remaining item in set X based on the CBF. Some false negative items will be found. Those false negative items are called the *potential false negative items* due to an incorrect item deletion operation.

In the following discussions, we measure the number of potential false negative items due to an incorrect item deletion, and then due to multiple incorrect item deletions respectively.

Lemma 1: The number of potential false negative items due to an incorrect item deletion (no incorrect item deletions have been performed before) is a discrete random variable, denoted as Y. Its possible values are the integers ranging from 1 to k.

Proof: Given any $x_1 \notin X$, its CBF address consists of counters $h_i(x_1)$ for $1 \leq i \leq k$, denoted as $\mathrm{Bfaddress}(x_1)$. Let us define a subset $X_i \subseteq X$ for each counter $h_i(x_1)$, where X_i contains the item $x \in X$ such that the CBF address $\mathrm{Bfaddress}(x)$ involves the counter $h_i(x)$. If the CBF occurs an incorrect deletion for the item x_1, the value of each counter $h_i(x_1)$, $1 \leq i \leq k$, is larger than 0 before the deletion operation, and is decreased by 1 after the deletion operation.

To expose the potential false negative items caused by the false deletion, let's delete an item of any subset X_i for $1 \leq i \leq k$ if each counter in its CBF address is > 0. We will thus decrease respective counters by one. Note that if the item is also in other subsets, it should be removed from those subsets. The CBF repeats the deletion operation until all counters of $h_i(x_1)$ for $1 \leq i \leq k$ are 0. By now, each subset X_i for $1 \leq i \leq k$ still contains one item (they may overlap). The reason is that, for the surviving item x of the subset X_i where $1 \leq i \leq k$, at least one counter of the $\mathrm{Bfaddress}(x)$ has been destroyed by the deletion of item x_1. Thus the deletion operation of the surviving x was taken over by the second item deletion rule, not the first one.

The cardinality of the union of the k subsets is indeed the number of false negative items caused by the incorrect deletion of x_1. It is a discrete random variable, and the possible values can been shown as follows. If the intersection of the k subsets is not empty initially, and the deletion operations of all the common items among k subsets have been taken over by the first kind of item deletion rule, the value of Y is k. If the deletion of one common item was taken over by the second kind of item deletion rule, then the value of Y is 1. Assume that the intersection of the k subsets is an empty set, but the intersection of $k-1$ subsets is not an empty set. If the deletion of one common item among $k-1$ subsets was taken over by the second kind of item deletion rule, then the value of Y is 2. And so on and so forth, the value of Y can be $3, 4, \ldots, k-1$. Thus, Lemma 1 holds. □

Theorem 1: Let i denote the possible values of the discrete random variable Y. The probability mass function of Y is

$$P(Y=i) = \frac{i! \sum_z \prod_{j=1}^{i} \binom{k}{a_j} \binom{k-\sum_{l=1}^{j-1} a_l}{a_j}}{\sum_{i=1}^{k} i! \sum_z \prod_{j=1}^{i} \binom{k}{a_j} \binom{k-\sum_{l=1}^{j-1} a_l}{a_j}}. \tag{2}$$

Proof: Assume i represents the possible value of Y, and all a_j for $1 \leq j \leq i$ are integers satisfying that $\sum_{j=1}^{i} a_j = k$. Let Y_i denote the event that Y=i, and means that i items are deleted due to the second item deletion rule and appeared as i false negative items. We need know the number of mutually exclusive outcomes which can produce the event Y_i.

Consider an experiment clustering the k bits to form i clusters, denoted as c_i. The integer k can be decomposed as the sum of i integers, and usually exists multiple different decomposition results. For each possible result,

1) The number of possible outcomes of the experiment is $\prod_{j=1}^{i} \binom{k-\sum_{l=1}^{j-1} a_l}{a_j}$.
2) Consider an experiment that establishes a bijective mapping between the number of i items and the i clusters. The number of possible outcomes is $i!$.
3) Based on the former steps, for a cluster c_i and its related item x, let us consider an experiment that establishes a bijective mapping between the CBF address $\mathrm{Bfaddress}(x)$ and the a_i bits of the cluster. The number of possible outcomes is $\binom{k}{a_i}$.

We then can calculate the number of possible outcomes of the experiment for each decomposition result, and it is

$$i! \prod_{j=1}^{i} \binom{k}{a_j} \binom{k - \sum_{l=1}^{j-1} a_l}{a_j}.$$

Let us calculate such value for other decomposition results of integer k using the same method. Then, the number of possible outcomes of the experiment to decomposing k as i clusters is

$$i! \sum_{\sum_{j=1}^{i} a_j = k} \prod_{j=1}^{i} \binom{k}{a_j} \binom{k - \sum_{l=1}^{j-1} a_l}{a_j}.$$

Let z denote that $\sum_{j=1}^{i} a_j = k$. According to the same method, we can calculate the number of possible outcomes for different i. Then the probability of Y_i is given by Formula 2. □

Corollary 1: The expectation of Y can be calculated by $E[Y] = \sum_{i=1}^{k} i \times P(Y = i)$.

Corollary 1 shows the expectation of the number of potential false negative items caused by an incorrect item deletion in theory. We find that some (or even all) potential false negative items are not exposed if respective counters in a CBF are still larger than 0 after the incorrect item deletion. On the other hand, the number of potential/exposed false negative items increases as the number of incorrect item deletion performed by the CBF increases.

Similar to the proof of Theorem 1, we have

Corollary 2: The number of cumulative potential false negatives caused by α undetectable incorrect item deletions is a discrete random variable. Its possible values are integers ranging from α to $\alpha \times k$. Its expectation is given by $\alpha \times E[Y]$.

3.2 Exposed false negative items in theory

In reality, we find that the potential false negative items caused by an incorrect item deletion often are not exposed simultaneously to future queries for remaining items in set X. The reason is that only one incorrect item deletion might not cause all affected counters reaching 0. The exposed false negative items show more realistic effect of an incorrect item deletion than the potential ones. We thus measure this new metric in theory in this section. Before in-depth analysis, we first introduce several definitions that will be used by later measurements.

Definition 1: Given a set X of n items and a random counter in a CBF, the events $A^n_{>i}$, $A^n_{<i}$ and $A^n_{=i}$ denote that the value of the related counter is larger than i, less than i, and equal to i respectively. The combination of the three events can produce new events $A^n_{\geq i}$ and $A^n_{\leq i}$.

The probability of the event $A^n_{=0}$ can be calculated by

$$P(A^n_{=0}) = (1 - 1/m)^{kn}.$$

The probability of the event $A^n_{=1}$ can be calculated by

$$P(A^n_{=1}) = \binom{kn}{1} \frac{(1 - 1/m)^{kn-1}}{m}.$$

The probability of the event $A^n_{\geq 1}$ can be calculated by

$$P(A^n_{\geq 1}) = 1 - P(A^n_{=0}).$$

We first study the probability that the potential false negative items, due to one and only one incorrect item deletion, are exposed or not. We then examine the expectation of the number of exposed false negative items caused by the incorrect deletion of one or multiple items.

Theorem 2: For an event that all potential false negative items, caused by an incorrect deletion of an item $x_1 \notin X$, are not exposed to queries for all items in the set X, its probability is

$$\left(1 - (1 - 1/m)^{kn-k}\right)^k = \left(P(A^{n-1}_{\geq 1})\right)^k. \tag{3}$$

Proof: If the CBF covers up all the potential false negative items after the incorrect deletion of an item x_1, all counters corresponding to $\mathrm{Bfaddress}(x_1)$ must be larger than 1 before deleting the item x_1. It is obvious that the construction process of the CBF is equivalent to throwing kn balls in m bins randomly. The event can be explained as follows. The bins $\mathrm{Bfaddress}(x_1)$ are put k balls such that each bin holds one ball. Other $kn{-}k$ balls are thrown in the m bins including the bins $\mathrm{Bfaddress}(x_1)$ randomly. The probability that each bin in $\mathrm{Bfaddress}(x_1)$ is hit at least once by the subsequent $k(n-1)$ balls is given by Formula 3. This finishes the proof. □

Theorem 3: For an event that a query for any item in the set X discovers one of potential false negative items caused by an incorrect deletion of an item $x_1 \notin X$, its probability is

$$\sum_{i=1}^{k} \binom{k}{i} P(A^{n-1}_{=0})^i P(A^{n-1}_{\geq 1})^{k-i} \Big(1 - \big(1 - \frac{i}{m \times P(A^n_{\geq 1})}\big)^k\Big). \tag{4}$$

Proof: According to the definition of this event, at least one counter among $\mathrm{Bfaddress}(x_1)$ is set to one and others are set to an integer larger than one. The event defined in this theorem can be explained as follows. First, let E_1

denote an event that exactly i counters among the counters $\mathrm{Bfaddress}(x_1)$ are set to one and other counters are set to some integers larger than one for $1 \le i \le k$. That is, each bin among the bins defined by $\mathrm{Bfaddress}(x_1)$ is put a ball firstly, and i bins among the bins $\mathrm{Bfaddress}(x_1)$ are not hit (other $k-i$ bins in $\mathrm{Bfaddress}(x_1)$ are hit by balls) during throwing $k(n-1)$ balls in the m bins. Its probability is

$$\binom{k}{i} P(A^{n-1}_{=0})^i P(A^{n-1}_{\ge 1})^{k-i}.$$

Second, on the base of E_1 let us consider another event E_2 that j hash functions hash the item x to the i counters that were set to one among the $\mathrm{Bfaddress}(x_1)$, and other $k-j$ hash functions map the item x to other counters set to nonzero among the m bits. We can infer that $1 \le j \le k$, and the probability of the event E_2 is

$$\begin{aligned} &\sum_{j=1}^{k} \binom{k}{j} \Big(\frac{i}{m \times P(A^n_{\ge 1})}\Big)^j \big(1 - \frac{i}{m \times P(A^n_{\ge 1})}\big)^{k-j} \\ = \;& 1 - \big(1 - \frac{i}{m \times P(A^n_{\ge 1})}\big)^k. \end{aligned}$$

Based on the conditional probability and total probability formulas, the probability that a subsequent set membership query will discover a false negative item can be calculated by Formula 4. This finishes the proof. □

Theorem 4: The number of false negative items, caused by an incorrect deletion and exposed to queries for all items in the set X, is a discrete random variable, denoted as Z. Its possible values are integers ranging from 1 to k. Its probability mass function is

$$P(Z = i) = \binom{k}{i} P(A^{n-1}_{=0})^i P(A^{n-1}_{\ge 1})^{k-i}. \qquad (5)$$

Proof: We assume that a false positive item $x_1 \notin X$ is deleted incorrectly from a CBF representing the set X with n items. Let E_1 denote an event that exactly i counters among the $\mathrm{Bfaddress}(x_1)$ are set to one and other counters are set to some integers larger than one. That is, each bin among the bins $\mathrm{Bfaddress}(x_1)$ is put a ball firstly, and i bins among the bins $\mathrm{Bfaddress}(x_1)$ are not hit (other $k-i$ bins in $\mathrm{Bfaddress}(x_1)$ are hit) during throwing $k(n-1)$ balls in the m bins. Thus, i counters among the $\mathrm{Bfaddress}(x_1)$ will become zero after deleting the item x_1 incorrectly from the CBF. According to the proof of Theorem 1, the i counters are mapped to i false negative items with high probability. On the other hand, the values of i range from 1 to k.

The probability of the event E_1 is

$$\binom{k}{i} P(A^{n-1}_{=0})^i P(A^{n-1}_{\ge 1})^{k-i}.$$

Thus, Formula 5 defines the probability mass function of Z. Theorem 4 is proved. □

Corollary 3: The expectation of Z is

$$E[Z] = \sum\nolimits_{i=1}^{k} i \times P(Z = i). \qquad (6)$$

The expectation of exposed false negative items due to α incorrect item deletions is $\alpha \times E[Z]$.

Proof: It is straightforward that the expectation of exposed false negative items due to one incorrect item deletion is given by Formula 6. Since the α incorrect item deletions are independent to each other, the cumulative value of exposed false negative items is $\alpha \times E[Z]$. □

3.3 Potential and exposed false negative items in practice

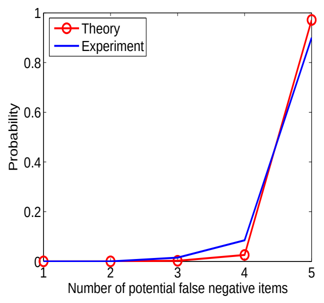
The hash functions are the fundamental factors which influence the distribution of the number of exposed false negative items. The ideal CBF makes the natural assumption that the hash functions can map each item in the unknown universe to a random number over the range $\{1, \ldots m\}$ *uniformly*. In reality, this assumption is too strict to achieve, thus it is very difficult to implement a CBF which can achieve the measurement results accurately mentioned above. Therefore, it is necessary to study the potential and exposed false negative items triggered by an incorrect item deletion from a practical aspect.

In the previous two subsections, the probability that any bit in a CBF is set to zero, one and an integer larger than one are studied analytically, just as many papers did. In reality, the method does not work well if the k hash functions can not satisfy the assumption of uniform random distribution. In such scenario, we use p_0, p_1 and p_2 to denote the fraction of bits set to zero, one and an integer larger than one in the CBF, and use them as the probability that any one bit is set to zero, one and an integer larger than one. The new method is reasonable from a viewpoint of the classic definition of probability, and also practical because it is easy to collect the p_0, p_1 and p_2. In this subsection, we reconsider the problems mentioned in the previous subsection from a practical aspect, and also revise the results along the following steps.

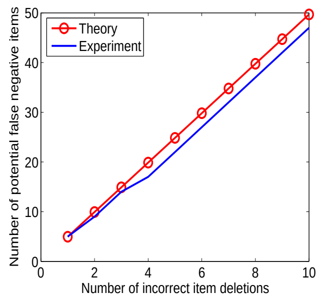
1) First of all, let us perform the following modifications that $P(A^n_{=0})$=p_0, $P(A^n_{=1})$=p_1, $P(A^n_{\ge 2})$=p_2 and $P(A^n_{\ge 1})$=$p_1 + p_2$.
2) The formulas using the $P(A^n_{=0})$, $P(A^n_{=1})$, $P(A^n_{\ge 2})$ and $P(A^n_{\ge 1})$ should perform the related modifications, such as Formula 3, 4, 5, and 6.

We further conduct experiments to verify theoretical results of potential and exposed false negative items caused by one or multiple incorrect item deletions. Specifically, we will verify the probability distribution of random variables Y and Z, the cumulative potential false negative items, and the cumulative exposed false negative items. We adopt the experimental methodology in Section 5.2 to design and implement experiments, where $c = 1$.

Figures 1(a) and 1(b) plot the theoretical and experimental results about the probability distribution of Y and the cumulative potential false negative items, respectively. The figures show that the experimental results match well with the theoretical results proved in Theorem 1 and Corollary 2. On the other hand, Figures 2(a) and 2(b) plot the theoretical and experimental results about the probability distribution of Z and the cumulative exposed false negative items, respectively. The figures show that the experimental results match well with the theoretical results proved in Theorem 4 and Corollary 3.

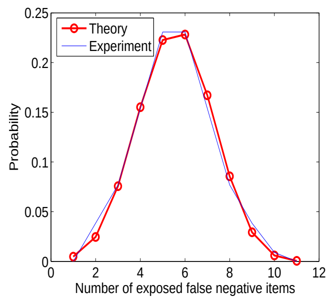


(a) The probability distribution of Y

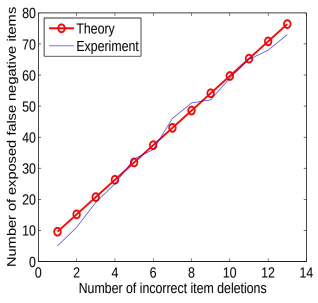


(b) The potential false negative items

Fig. 1. The provability distribution and number of potential false negative items in theory as well as experiment, where m=1600, n=100, and k=5.



(a) The probability distribution of Z



(b) The exposed false negative items

Fig. 2. The probability distribution and number of exposed false negative items in theory as well as experiment, where m=1600, n=100, and k=11.

Thus, the experimental results verify the correctness of those theoretical results.

By comparing Figure 1(a) with Figure 2(a), we can find that one incorrect item deletion can cause k potential false negative items with high probability, however, only about $k/2$ exposed false negative items with high probability. This motivates us to study the exposed false negative items besides the potential ones.

3.4 Principles to improve counting Bloom filter

According to the measurement results and observations mentioned above, we find two useful and important principles to improve counting Bloom filter.

1) The improved CBF should not increase the probability of false positive. Thus, at least it should not decrease the ratio of bits set to zero, p_0.
2) The improved CBF should decrease the exposed false negative items caused by an incorrect item deletion operation. Thus, at least it should increase the ratio of bits set to a value larger than one, p_2.

To increase p_2, more bits of the CBF address of an item $x_1 \notin X$ need to be set to an integer larger than one. The activity will decrease the number of exposed false negative items, caused by the incorrect item deletion operation triggered by x_1. The two principles motivate us to consider a possible improvement of CBF to increase p_2 but do not decrease p_0. In the next section, we propose a new mechanism to improve CBF and achieve the desired objectives. Although the solution may not be the best one, it is useful to reduce the exposed false negative items.

4 Multi-choice Counting Bloom Filter

The CBF assigns each item $x \in X$ just one CBF address $\text{Bfaddress}(x)$. The addresses of different items are independent each other, and the value distribution of all bits in a CBF is uncontrollable. Therefore, it is very difficult to increase p_2 by the traditional mechanisms widely used to control the probability of false positive, for example the parameter optimization. If we introduce some choices about the CBF address, it is possible to increase p_2 with the help of a suitable greedy algorithm. In this paper, we will combine the CBF with a mechanism often applied to improve load balancing, the power of more choices [26], and validate the two principles mentioned above.

Lumetta et al. combine the power of two choices with Bloom filter to reduce the false positive probability [18]. They use two groups of hash functions for mapping items and checking membership at the cost of additional computations. Their experiments show that the solution does not decrease the false positive probability under *any* configuration of the parameters m, n, and k in the online model, but achieves some improvement in the offline model. Recently, Jimeno et al. propose a Best-of-N Bloom filter replacing two groups of hash functions with N groups [19]. They show the idea works well under major configurations in the online model, and the increase of N always decreases the probability of false positive judgment.

The idea of our solution is similar to that of [18], [19], but the objective and related methods are different. We try to decrease the exposed false negative items triggered by an incorrect item deletion, but the authors of [18], [19] aimed to control the false positive probability and did not mention any issue about the false negative judgment. We propose a more suitable item insertion method to realize our objective, which increases the fraction of bits set to an integer larger than one and does not decrease the fraction of bits set to zero in the filter. The idea of using more choices to improve CBF should also support the item deletion operation, however, this problem is not discussed in [18], [19]. We propose a reasonable solution to handle this issue, and analyze its impact on the false positive and false negative judgments. The average time complexities of the following item operations for CBF and MCBF are $O(k)$ and $O(c \times k)$, respectively. The space complexities for CBF an MCBF are the same, $L \times m$.

4.1 Insertion operation

Given a dynamic data set X with n items and a CBF with a vector of m bits, let us consider the following variation on the CBF. Instead of using a group of hash functions to assign a CBF address for any item $x \in X$, we use c groups of hash functions to produce c CBF addresses as candidates where $c \geq 2$. The ith group consists of k hash functions $h_1^i, h_2^i, \cdots, h_k^i$ for $1 \leq i \leq c$, and produces the ith CBF address for the item x, denoted as $\text{Bfaddress}^i(x)$. We require that hash functions are perfectly independent, and as random as possible. We call the improved CBF as multi-choice counting Bloom filter, abbreviated as MCBF.

After receiving an insertion request for an item $x \in X$, the MCBF will first calculate c CBF addresses for x as the candidates, then choose one as the final CBF address according to different approaches, finally increase the value of counter by one for each bit in the final CBF address. A natural greedy approach is proposed in literature [18], [19]. The basic idea is to calculate how many additional bits would have to be set to one for each candidate and select the candidate with the least additional bits to be set to one. The greedy approach produces less number of bits set to nonzero than that produced by the CBF. It is well known that decreasing the number of bits set to nonzero in a filter will increase the number of bits set to an integer larger than one in the same filter. Thus, the greedy approach also has positive impact on implementing our fundamental objective although its original goal is to decrease the false positive probability.

In this paper, we propose an improved greedy approach. It still increases the ratio of the bits whose value is zero at the similar extent, just as the greedy approach does. At the same time, it can increase the ratio of the bits whose value is larger than one and decrease the ratio of the bits set to one than that of the greedy approach. Thus, our new approach can decrease the exposed false negative items, caused by an incorrect item deletion, more than what the greedy approach does. Algorithm 1 explains the improved greedy approach in detail.

Algorithm 1 Improved Greedy Approach For Selecting a CBF Address for an item x

```
1: Define and calculate a metric for each candidate to measure the number of additional bits needed to be set to one in order to cover $x$. Then select the minimum value. If there is only one candidate whose metric value equals to the minimum value, then that candidate is the final CBF address of $x$. Otherwise, go to the next step with the candidates whose metric equals to the minimum value as the input parameters.
2: Define and calculate a metric for each candidate contained in the input parameters to measure the number of bits set to one currently. Then pick up the maximum value. If there is only one candidate whose metric value equals to the maximum value, then that candidate is the final CBF address of $x$. Otherwise, go to the next step with the candidates whose metric equals to the maximum value as input parameters.
3: Define and calculate a metric for each candidate contained in the input parameters to measure the maximum value among the $k$ counters. Then pick up the minimum value. If there is only one candidate whose metric value equals to the minimum value, then it is the final CBF address of $x$. Otherwise, randomly select one from the candidates whose metric equals to the minimum value as the final CBF address of $x$.
```

4.2 Query operation

When we query an item x, we will compute the $\mathrm{Bfaddress}^i(x)$ for the ith group of hash functions, for $i \in [1, c]$, and test whether all bits of $\mathrm{Bfaddress}^i(x)$ are non-zero. If it is, then we say x passes the test of ith group. The query for x returns "yes" if it passes the test of *any* group $i \in [1, c]$.

Both the greedy approach and the improved greedy approach can increase the ratio of bits set zero, and seem to decrease the chances of a false positive. But, using c groups of hash functions would seem to increase the chances of a false positive, in that there are now c ways for a false positive to occur. Specifically, recall that p_0 is the fraction of bits set to zero in the filter, the probability of a false positive is

$$1 - (1 - (1 - p_0)^k)^c. \tag{7}$$

It is very difficult to determine whether the greedy and improved approaches always decrease the false positive probability in the online model, although some preliminary analysis have been done in literatures [18], [19]. Lumetta and Mitzenmacher did not provide clear comments about the problem. Jimeno and Christensen believed that the greedy approach always decreases the false positive probability as the increasing of c. Recall that the strict assumptions about hash functions are very difficult to reach, thus the analysis based on such assumptions can not reflect the reality. In this paper, we prefer to use the experiments rather than the theoretical analysis to address the problem. The results of our experiments show that the greedy and improved greedy approaches decrease the false positive probability as the increasing of c when the ratio m/n exceeds a threshold, but increase it as the increasing of c under other configurations.

4.3 Deletion operation

During the process of item deletion, an item $x \in X$ may find multiple possible CBF addresses. It is clear that only one CBF address is assigned to the item x during the insertion process, and others are false positive judgments. In such situation, if the MCBF persists in performing the item deletion operation, the related counters of a wrong CBF address may be decreased by one with some probability. As discussed in Section 2.3, the *incorrect deletion of a multi-address item* always destroys the CBF, and produces at most k potential false negative items.

After the representation of set X, let E_{multi} denote the event that an item $x \in X$ meets multiple possible CBF addresses when performing the membership query of item x. The event means that at least an additional group of hash functions map the item x to the bits set to an integer larger than zero besides the group of hash functions related to the real CBF address of item x. The probability of the event is

$$P(E_{multi}) = 1 - (1 - (1 - p_0)^k)^{c-1}. \tag{8}$$

Formula 8 yields an upper bound on the probability that event E_{multi} happens, and an upper bound on the number of multi-address items is $n_r \times P(E_{multi})$, where n_r denotes the cardinality of set X. Our experimental results show that the real number of such items is much less than the upper bound $n_r \times P(E_{multi})$. On the other hand, the deletion of a such item is not always performed incorrectly in related CBF. That is, the deletion of a multi-address item also has chance to be performed correctly without causing false negative items. Theorem 5 analyzes the problem from the probability theory.

According to the definition of event E_{multi}, the number of false positives among the possible CBF addresses of item x is a discrete random variable, which is denoted as U.

Theorem 5: For an item x which meets an event E_{multi}, the probability that the deletion of the item x produces false negative items is

$$\frac{E[U]}{1+E[U]}. \tag{9}$$

Proof: The set of possible values of variable U are the integers ranging from 1 to $c-1$. Its probability mass function is

$$P(U=i)=\binom{c}{i}\Big((1-p_0)^k\Big)^i\Big(1-(1-p_0)^k\Big)^{c-i}.$$

Furthermore, $E[U]=\sum_{i=1}^{c-1} i\times P(U=i)$ denotes the expectation of U. Therefore, the number of possible CBF addresses of x is $1+E[U]$, and the probability that the deletion of x will cause false negative items is $\frac{E[U]}{1+E[U]}$. This finishes the proof. □

According to Theorem 5 and the method to measure the number of potential and exposed false negative items, we estimate an upper bound on the number of potential and exposed false negative items if the CBF deletes one of such kind of items. We can also calculate upper bounds on the number of potential false negative items and that of exposed false negative items caused by all multi-address items. In reality, the number of potential and exposed false negative items are much less than the related upper bounds, because the number of such items is much less than its upper bound.

Traditionally, a MCBF decrements the values of the respective counters when it receives a deletion instruct of a multi-address item. As a new policy, a MCBF can simply omit the deletion instruct of a multi-address item. The objective is to prevent the MCBF from producing potential and exposed false negative items. Thus, the MCBF still keeps membership information for at most $n_r \times P(E_{multi})$ items even it receives the deletion instructions for all items of the set X. If other items join the set X during the process of deleting the original items, the MCBF reflects not only current items of the set X but also at most $n_r \times P(E_{multi})$ retained items. It is reasonable that the false positive probability of the MCBF is always larger than the theoretical value. But, the difference between the real value and theoretical value is small, and the negative impact of the new policy can be controlled at an acceptable level. As direct results of the new policy, queries of such items always response false positive, and the filter does not need to do any change when such items rejoin the set X.

In summary, if a MCBF deletes multi-address items, it may result in false negative items, otherwise it increases the false positive probability. In reality, it needs to make a tradeoff between these two policies. For applications in which the harm of false negative is more serious than that of false positive, it's better to keep the MCBF after receiving a deletion request for a multi-address item. On the other hand, it's better to do related modifications to the filter. Recall that even keep all multi-address items in a MCBF, the negative impact on the false positive probability can be controlled at a low and acceptable level. We therefore recommend to keep multi-address items in filter because the harm of false negative items is serious for major applications.

5 Performance Evaluations

We first describe the implementation issues of related CBF and the configurations of our experiments, and then compare the analytical model with the experiment results in terms of exposed false negatives. We also evaluate the false positive probability, the greedy and improved greedy insertion of items, and the impact of item deletion methods.

5.1 Implementation

In this work, we extend the BF and CBF delivered by Guo et al. in [17] to implement the multi-choice counting Bloom filter. One critical factor of the multi-choice counting Bloom filter is to create c groups of hash functions. In our experiments, a group of k hash functions are generated by

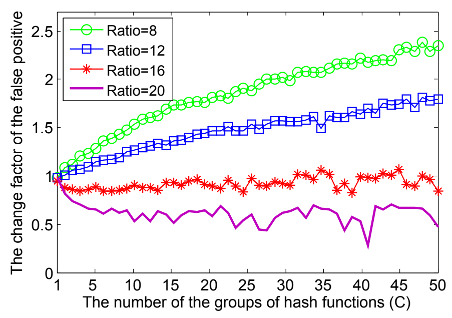
$$h_i(x)=\big(g_1(x)+i\times g_2(x)\big) \bmod m, \tag{10}$$

where $g_1(x)$ and $g_2(x)$ are two independent and random integers in the universe with range $\{1,2,\ldots,m\}$. The value of i ranges from 0 to $k-1$. We propose the following three methods to generate two random integers for any item x.

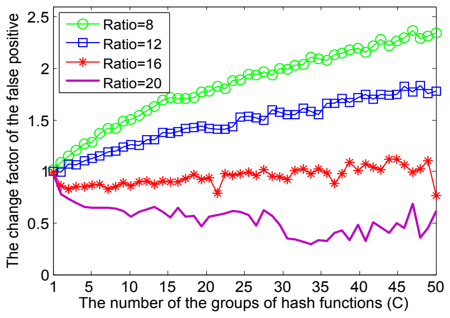
1) The SDBM_BUZhash method. We choose the SDBM and BUZ hash functions to produce the values of $g_1(x)$ and $g_2(x)$, respectively.
2) The SDBM_MersenneTwister method. The output of SDBM Hash function acts as the seed of a random number generator (RNG) MersenneTwister. The MersenneTwister produces two desired random integers.
3) The BUZ_MersenneTwister method. The output of BUZ hash function acts as the seed of MersenneTwister. The MersenneTwister produces two desired random integers.

The mechanism requires two hash functions or one hash function and one random number generator to run k rounds of Formula 10 in order to generate a $\mathrm{Bfaddress}(x)$ for an item x. For other $c-1$ Bloom filter addresses, we use the results of appending $c-1$ predefined strings on x as the inputs for producing $c-1$ pairs of two random numbers, and then achieve other $c-1$ CBF addresses by the Formula 10. The mechanism can bring in a considerable reduction in processing overhead compared to using $c\times k$ hashes, and does not increase the false positive probability [27].

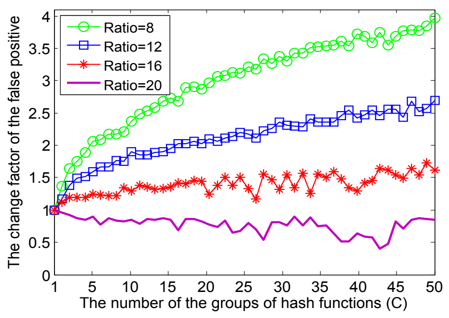
The quality of the hash functions and one random number generator has significant impact on the experiment results. The SDBM hash function has a good overall distribution for different data sets, and works well even if the MSBs of items in a data set exhibit high variation. BUZ hash function is fast and employed widely. It produces near-perfect result even with extremely skewed input data. The Mersenne twister provides for fast generation of very high quality pseudo-random numbers, and is designed to rectify many flaws found in older algorithms.



(a) $SDBM\_MersenneTwister$ method

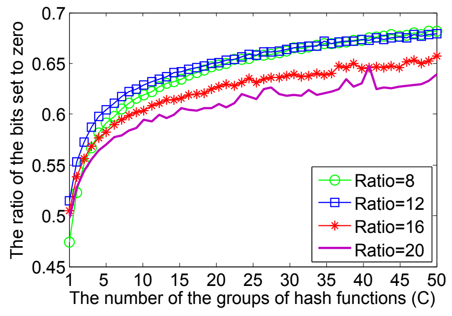


(b) $BUZ\_MersenneTwister$ method

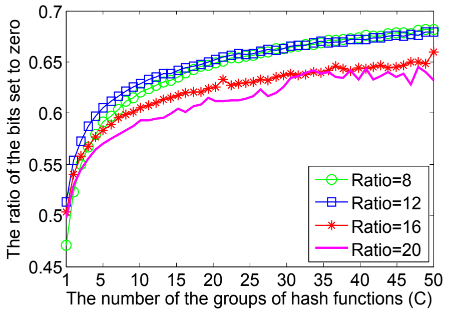


(c) $SDBM\_BUZhash$ method

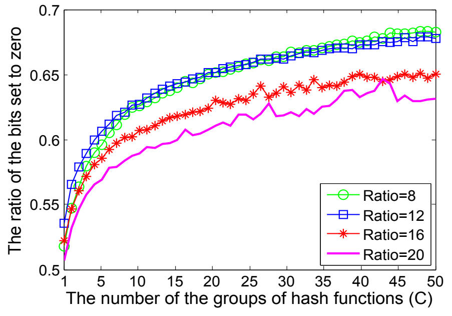
Fig. 3. The ratio of false positive probability of multi-choice counting Bloom filter to that of Bloom filter.



(a) $SDBM\_MersenneTwister$ method



(b) $BUZ\_MersenneTwister$ method



(c) $SDBM\_BUZhash$ method

Fig. 4. The ratio of the bits set to zero in multi-choice counting Bloom filters with different configurations.

5.2 Experiment methodology

Note that CBF and MCBF are designed to represent any possible sets, query sequences, and item deletion/insertion sequences. In addition, there are no benchmark sets and traces in the field of Bloom filters. Since we could not obtain traffic traces [28] in the field of real-time identification of P2P traffic based on CBF, we simply use a set from the DBLP. The size of the data set is near 300M. We retrieve partial history information of papers published in the major conferences from the DBLP records. We then use the name of authors to initialize a data set $\mathcal{X}$ to be represented by our Bloom filter, and another data set $\mathcal{Y}$ to be used by the tests of false positive judgments. Our experiments do not seek particular sequences of item query/correct deletion/incorrect deletion/insertion but simply use a synthetic random sequence. The limitation is that we did not use the actual traces. We plan to work on the real traces once we obtain them. While the extension is necessary in a deeper, trace-drive study, the initial results are independent to the type of set and the sequence of those basic item operations facing MCBF.

For each instance of experiment, we initialize the following parameters before testing data. The first parameter is the bits per item $ratio = m/n$, and can be set as 8, 12, 16, 20 and 24. The second parameter k is $\lfloor (m/n)\ln 2 \rfloor$ [1]. The third parameter n is set to 10000. The fourth parameter is the upper bound of c, and is set to 50. The fifth parameter T is the size of data set used by the tests of false positive judgments, and is set to $8 \times n$. The hash algorithms are the three candidates mentioned above. The item insertion algorithms are the greedy together with its improved algorithms.

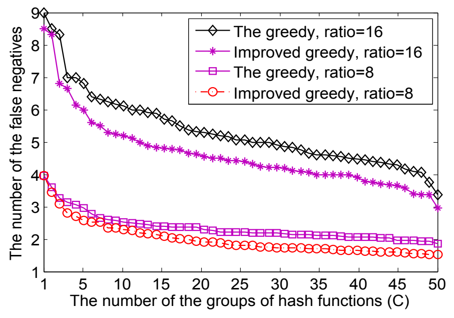
The experiments are divided as $3 \times 2 \times 5 = 30$ instances. Each instance selects one hash algorithm from three candidates, one item insertion algorithm from two candidates, and the value of m/n from five candidates. Other parameters are the same among different instances. Each instance runs c rounds, with one round for each integer in the range $[1, c]$. The 30 instances are conducted on a cluster with Linux and Solaris OS and more than 30 CPUs.

5.3 Experiment results

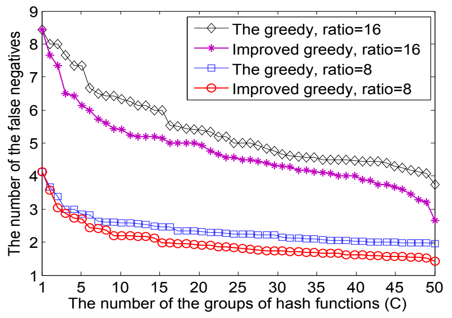
5.3.1 False positive judgment

Figures 3 and 4 plot the experiment results about the false positive probability under several different configurations, and we report results under the improved greedy algorithm only. The results under the greedy algorithm are similar, and are omitted due to the space limit. The *change factor* of false positive is the ratio of false positive probability of MCBF to that of CBF. Figure 4 shows that MCBF always increases the ratio of the bits set to zero as the increasing of c, and the number of bits set zero in a MCBF with c=50 increases about 40% of that in a CBF when m/n=8 or 12. The gain is about 30% when m/n=16 or 20. The results demonstrate that when MCBF satisfies the first policy we proposed in section 3.4, the ratio of bits set to zero increases significantly and the false positive probability might be decreased.

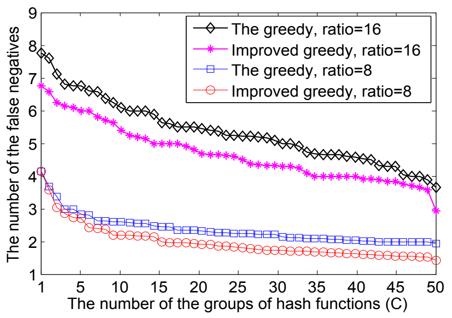
For the cases that $m/n = 8$ or 12, the false positive probability of MCBF is always larger than that of CBF as the increasing of c. For the case that $m/n = 20$, MCBF indeed decreases the false positive probability as the increasing of c. The experiment results under the three different



(a) $SDBM\_MersenneTwister$ method

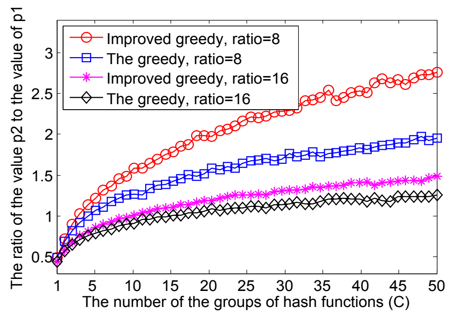


(b) $BUZ\_MersenneTwister$ method

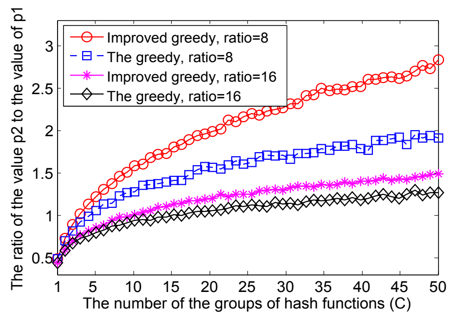


(c) $SDBM\_BUZhash$ method

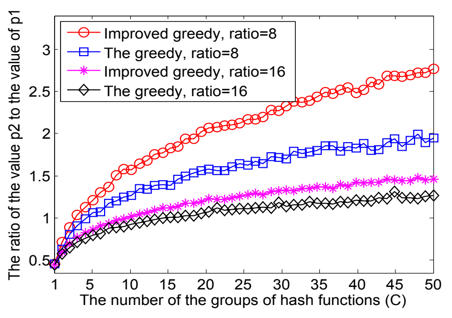
Fig. 5. The average value of exposed false negative items due to an incorrect item deletion triggered by a false positive.



(a) $SDBM\_MersenneTwister$ method



(b) $BUZ\_MersenneTwister$ method



(c) $SDBM\_BUZhash$ method

Fig. 6. The ratio of the bits set to a value larger than one to the bits set to one.

hash algorithms have the similar trend for the four cases of m/n. There may exist a threshold of the value of m/n such that the false positive probability of MCBF always decreases as the increasing of c only if m/n exceeds it. When $m/n = 16$, the hash algorithm is SDBM_MersenneTwister or BUZ_MersenneTwister, the false positive probability of MCBF is less than or similar to that of CBF as the increasing of c. If the hash algorithm is the SDBM_BUZhash, the false positive probability of MCBF is always larger than that of CBF as the increasing of c. The experiment results show that the SDBM_MersenneTwister and BUZ_MersenneTwister are more suitable to the multi-choice counting Bloom filters than the SDBM_BUZhash.

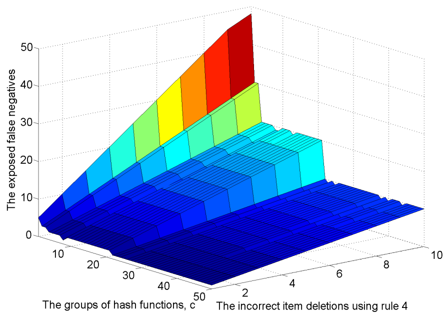
In summary, the results show that MCBF satisfies the first policy used to improve CBF, however, it cannot always decrease the false positive probability in the online model under any configurations. The reason is that the positive contribution by increasing the ratio of the bits set to zero does not always go beyond the negative influence of the more chance of a false positive resulting from the c possible Bloom filter addresses. Thus, it is very important to tune the parameters of MCBF carefully in order to always decrease the false positive probability. The result recommends that the value of m/n should not less than 16 and prefers the SDBM_MersenneTwister and BUZ_MersenneTwister hash algorithms.

5.3.2 *False negative judgment*

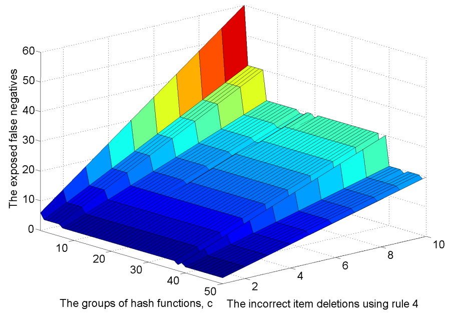
Theoretically, we show that MCBF can reduce the exposed false negatives caused by incorrect deletion of items. Recall that the experiment are divided into 30 instances. Now we examine whether the experiment result is consistent with theoretical result in each instance. The incorrect item deletions triggered by different false positives have different impacts on the exposed false negative items. Due to the huge number of possible false positives in a given MCBF, here we only show two representative categories.

In the first category, we emulate an incorrect deletion of an item by decreasing the counters of k bits by one, where the k bits are randomly selected from those bits set to nonzero in the MCBF. After multiple rounds of each instance, the average value of the number of exposed false negative items due to an incorrect item deletion is shown in Figure 5. As the analysis in theory, the improved greedy algorithm indeed decreases the exposed false negatives more than the traditional greedy algorithm under different configurations of m/n and hash algorithms. We use the experiment results shown in Figure 6 to explain the reason of such conclusion. In Figure 6, a curve of our improved greedy algorithm is always above a corresponding curve of the traditional greedy algorithm in each experiment instance. This means that our improved greedy algorithm updates more bits set to one with a value larger than one than the traditional greedy algorithm. Thus, more potential false negative items can be covered and are not exposed under the improved greedy algorithm. In summary, the improved greedy algorithm outperforms the traditional one in theory and practice.

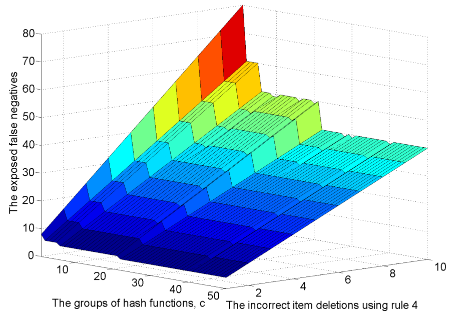
In the second category, we emulate an incorrect deletion of an item by decreasing any α counters set to one and β counters set to an integer larger than one by one in the MCBF, where $k = \alpha + \beta$ and $\frac{\alpha}{\beta} \approx \frac{p_1}{p_2}$. This method can reflect an incorrect deletion of an item more accurate than the method used in the first category. We then measure



(a) $SDBM\_MersenneTwister$ method, ratio=8

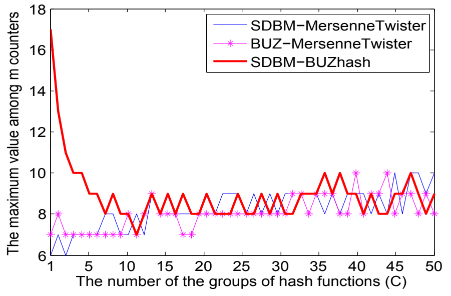


(b) $BUZ\_MersenneTwister$ method, ratio=12

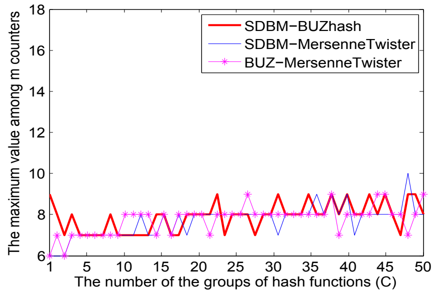


(c) $SDBM\_BUZhash$ method, ratio=16

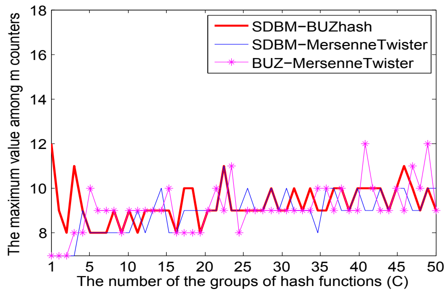
Fig. 7. The exposed false negative items due to multiple incorrect item deletions triggered by multiple false positives.



(a) $Ratio = 8$

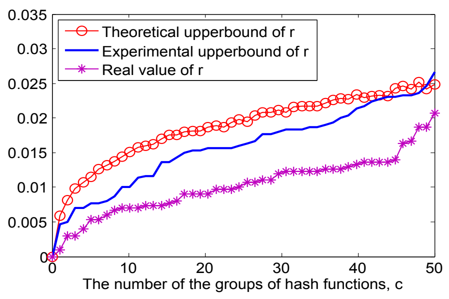


(b) $Ratio = 12$

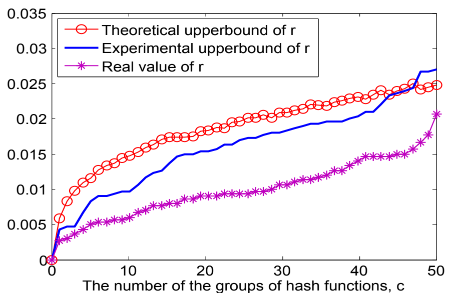


(c) $Ratio = 16$

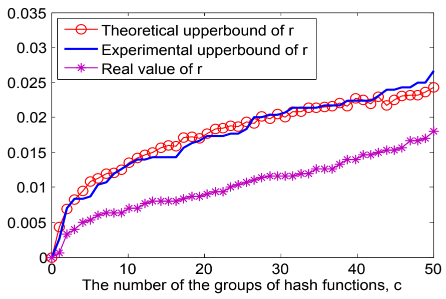
Fig. 8. The maximum counter value among all the m bits in the context of our improved greedy algorithm.



(a) $SDBM\_MersenneTwister$, ratio=8



(b) $BUZ\_MersenneTwister$, ratio=8



(c) $SDBM\_BUZhash$, ratio=8

Fig. 9. The ratio of items with multiple addresses and the false positive probability.

the exposed false negative items due to one incorrect item deletion as well as the cumulative exposed false negative items due to multiple incorrect item deletions. Indeed, this set of experiments covers the scope of the experiments in the first category. The experiment results under the improved greedy algorithm is shown in Figure 7. We find that the number of exposed false negative items increases more as the increasing of number of incorrect item deletions in related CBF, and at least 50% of the exposed false negative items become unexposed if we introduce a MCBF with at most 4 groups of hash functions when $ratio$=8. When $ratio$=12 and $ratio$=16, a MCBF needs 10 and 20 groups of hash functions, respectively, to achieve the similar result. The MCBF also makes 80% of the exposed false negative items in a CBF become unexposed by assigning a moderate value to the parameter c. The results show that our improved greedy algorithm still do better than the traditional one in this scenario. We do not show the detailed results due to the page limit. The similar results hold when $ratio$=20 and $ratio$=24 in both first as well as second category, and are omitted.

In summary, the results indicate that MCBF satisfies the second policy to improve CBF mentioned in subsection 3.4 and makes about $50-80\%$ of exposed false negative items in a CBF become unexposed with the help of careful configuration. It, however, does not mean that c should be as large as possible because of the additional computation costs. On the other hand, the contributions of decreasing the false positive probability and reducing the number of exposed false negative items turns to be trivial after c exceeds a certain threshold in MCBF.

5.3.3 The maximum load

Recall that each array position of a CBF is allocated L bits, and L=4 suffices if the k hash functions can map each item over the range $1, ..., m$ uniformly and independently. Under the context of multi-choice counting Bloom filter, obviously, this assumption is not true. Hence, it is necessary to reconsider whether L=4 still suffices. We conduct 9 experiment instances to achieve the maximum load among the m array positions under different configurations. Each instance selects one hash

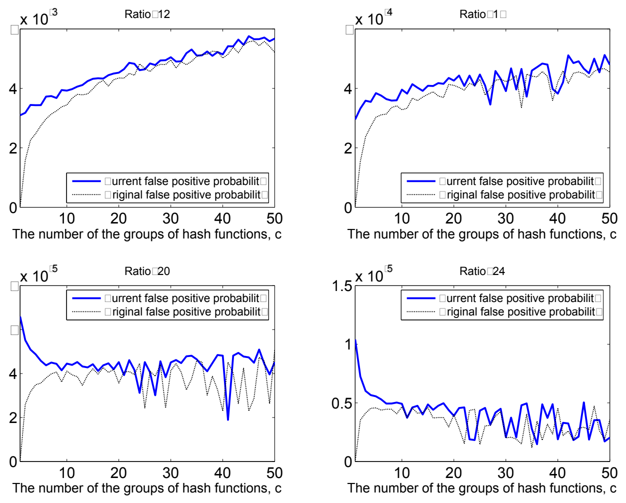


Fig. 10. The original false positive probability and the resulting false positive probability due to keep additional r items.

function algorithm from three candidates, and the ratio denoted the value of m/n from 8,12, and 16. Other parameters are the same among different instances.

The experimental results shown in Figure 8 indicates the maximum load is less than 16 in 8 instances except one instance using the $SDBM\_BUZhash$ algorithm. This shows that other two hash algorithms are more suitable to the MCBF than the $SDBM\_BUZhash$. Recall that the same conclusion has been proposed in subsection 5.3.1. The experimental results also show that the maximum load is less than 16 when $m/n \geq 16$ which we do not show. On the other hand, the CBF and theirs variations often assign 8 or a larger value to the parameter m/n in order to decrease the false positive probability. In summary, L=4 still suffices in the context of the MCBF.

5.3.4 Impact of item deletion operation in MCBF

Recall that some items of a set X may have multiple possible CBF addresses in a MCBF. The recommended item deletion operation mentioned in 4.3 remains the set membership information of such items when dealing with item deletion requests. It is easy to know that this operation may increase the false positive probability of the MCBF. Therefore, the number of such items should be as less as possible. Let r denote the percentage of such items in the X. An estimated upper bound of r is given by Formula 8 in theory. Each experiment instance calculates the number of such items in the X, and then achieves the experimental upper bound of r. We also measure another metric called the real value of r besides the theoretical and experimental upper bounds of r. Each instance attempts to delete all items from the X and related MCBF following the recommended deletion operation. Finally, the ratio of remaining items to the original items in the X denotes the real value of r.

Figure 9 shows that the real value of r is always less than its estimated and experimental upper bounds. In Figures 9(a), 9(b), and 9(c), all the three curves increase as the increasing of c. The curves of the experimental upper bound of r and the real value of r increase smoothly, and remain at a low level. In practice, the frequency of deleting all items from a data set and its MCBF is very low. If we insert other n items to the data set and related MCBF once this event happens, the false positive probability of resulting MCBF is often larger than that of the original one, but is still at a lower and stable level. Figure 10 shows that it is appropriate to the original value when $c \geq 10$. In summary, the item deletion operation can avoid producing false negative items at the cost of a trivial influence on the false positive judgment if the size of the X changes at a stable level without immediately decreasing more.

6 Conclusions

We show that the false negative items can indeed occur in a CBF and related variants. We also reveal that two types of incorrect item deletion operations triggered by a false positive are the root causes of false negative items, and the potential false negative items usually are not fully exposed at the same time. We then measure the potential and exposed false negative items from aspects of theory and practice. Finally, we introduce two fundamental principles to make more potential false negative items become unexposed whenever possible, and propose an improved CBF to validate our principles. Our analytical and experimental results demonstrate that the proposed CBF decreases the number of exposed false negative items without increasing the probability of false positive.

Acknowledgments

The authors would like to thank anonymous reviewers for their constructive comments. The work of Deke Guo is supported in part by the NSF China under grants No. 60903206 and No. 60903225. Xiang-Yang Li and Yunhao Liu's research is supported in part by the NSF CNS-0832120, the NSF China under grants No. 60828003, No. 60773042 and No. 60803126, the Natural Science Foundation of Zhejiang Province under grant No. Z1080979, the National Basic Research Program of China (973 Program) under grant No. 2010CB328100 and No. 2006CB30300, the National High Technology Research and Development Program of China (863 Program) under grant No. 2007AA01Z180, the Hong Kong RGC under grant HKBU 2104/06E, and the CERG under Grant PolyU-5232/07E. The work of Panlong Yang is supported in part by the 973 Program of China under grant No. 2009CB3020402, and the 863 Program of China under grant No. 2008AA01Z216.

References

[1] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[2] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2005.

[3] J. K. Mullin. Optimal semijoins for distributed database systems. *IEEE Trans. Software Eng.*, 16(5):558–560, 1990.

[4] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Networking*, 8(3):281–293, 2000.

[5] J. Li, J. Taylor, L. Serban, and M.Seltzer. Self-organization in peer-to-peer system. In *Proc. 10th ACM SIGOPS European Workshop*, Saint-Emilion, France, September 2002.

[6] S. C. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *Proc. IEEE INFOCOM*, pages 1248–1257, New York, USA, June 2004.

[7] A. Kumar, J. Xu, and E. W. Zegura. Effcient and scalable query routing for unstructured peer-to-peer networks. In *Proc. IEEE INFOCOM*, pages 1162–1173, Miami, USA, March 2005.

[8] F. Deng and D. Rafiei. Approximately detecting duplicates for streaming data using stable bloom filters. In *Proc. 25th ACM SIGMOD*, pages 25–36, Chicago, Illinois, USA, June 2006.

[9] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese. Beyond bloom filters: from approximate membership checksto approximate state machines. In *Proc. ACM SIGCOMM*, pages 315–326, Pisa, Italy, September 2006.

[10] Kang Li and Zhenyu Zhong. Fast statistical spam filter by approximate classifications. In *Proc. SIGMETRICS/Performance*, pages 347–358, Saint Malo, France, June 2006.

[11] M. Mitzenmacher. Compressed bloom filters. *IEEE/ACM Trans. Networking*, 10(5):604–612, 2002.

[12] A. Kirsch and M. Mitzenmacher. Distance-sensitive bloom filters. In *Proc. 8th Workshop on Algorithm Engineering and Experiments (ALENEX06)*, Miami, Florida,USA, January 2006.

[13] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li. Space-code bloom filter for efficient per-flow traffic measurement. In *Proc. 23th IEEE INFOCOM*, pages 1762–1773, Hongkong, China, March 2004.

[14] S. Cohen and Y. Matias. Spectral bloom filters. In *Proc. 22th ACM SIGMOD*, pages 241–252, San Diego, USA, June 2003.

[15] R. P. Laufer, P. B. Velloso, and O. C. M. B. Duarte. Generalized bloom filters. Technical Report Research Report GTA-05-43, University of California, Los Angeles (UCLA), September 2005.

[16] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal. The bloomier filter: an efficient data structure for static support lookup tables. In *Proc. 5th SODA*, pages 30–39, New Orleans, Louisiana, USA, January 2004.

[17] D. Guo, J. Wu, H. Chen, and X. Luo. Theory and network applications of dynamic bloom filters. In *Proc. 25th IEEE INFOCOM*, Barcelona, Spain, April 2006.

[18] S. Lumetta and M. Mitzenmacher. Using the power of two choices to improve bloom filters. http://www.eecs.harvard.edu/michaelm/postscripts/.

[19] M. Jimeno, K. Christensen, and A. Roginsky. A power management proxy with a new best-of-n bloom filter design to reduce false positives. In *Proc. 26th IEEE International Performance Computing and Communications Conference (IPCCC)*, Louisiana, USA, April 2007.

[20] P. S. Almeida, C. Baquero, N. M. Preguiça, and D. Hutchison. Scalable bloom filters. *Inf. Process. Lett.*, 101(6):255–261, 2007.

[21] D. Benoit, B. Bruno, and F. Timur. Retouched bloom filters: Allowing networked applications to trade off selected false positives against false negatives. In *Proc. ACM CoNEXT*, Lisboa, Portugal, September 2006.

[22] Y. Zhu and H. Jiang. False rate analysis of bloom filter replicas in distributed systems. In *Proc. 35th International Conference on Parallel Processing (ICPP)*, pages 255–262, Ohio, USA, August 2006.

[23] D. Forsgren, U. Jennehag, and P. Osterberg. Objective end-to-end QoS gain from packet prioritization and layering in MPEG-2 streaming video. http://amp.ece.cmu.edu/packetvideo2002/papers/61-ananhseors.pdf.

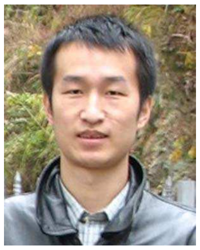
[24] T. Karargiannis, A. Broido, M. Faloutsos, and K. C. Claffy. Transport layer identification of P2P traffic. In *Proc. ACM SIGCOMM*, 2004.

[25] K. Thomson, G. J. Miller, and R. Wilder. Wide-are traffic patterns and characteristics. *IEEE Network*, December 1997.

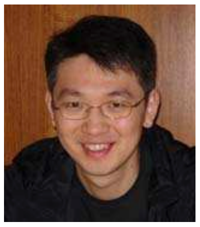
[26] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Syst.*, 12(10):1094–1104, 2001.

[27] A. Kirsch and M. Mitzenmacher. Less hashing, same performance: Building a better bloom filter. In *Proc. 14th Annual European Symposium on Algorithms*, pages 456–467, Switzerland, 2006.

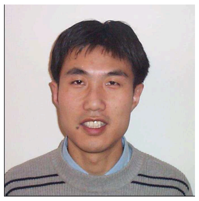
[28] T. Karargiannis, A. Broido, M. Faloutsos, and K.C. Claffy. Transport layer identification of p2p traffic. In *Proc. ACM SIGCOMM*, September 2004.



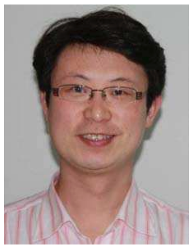
Deke Guo received the B.S. degree in industry engineering from Beijing University of Aeronautic and Astronautic, Beijing, China, in 2001, and the Ph.D. degree in management science and engineering from National University of Defense Technology, Changsha, China, in 2008. He was a visiting scholar at the Department of Computer Science and Engineering in Hong Kong University of Science and Technology from Jan. 2007 to Jan. 2009. He is now an assistant professor of Information System and Management, National University of Defense Technology, Changsha, China. His current research interests include peer-to-peer computing, Bloom filters, data center networking, and wireless networks. He is a member of the ACM and the IEEE.



Yunhao Liu (SM'06) received the B.S. degree in automation from Tsinghua University, China, in 1995, and the M.A. degree from the Beijing Foreign Studies University, China, in 1997, and the M.S. and Ph.D. degrees in computer science and engineering from Michigan State University in 2003 and 2004, respectively. He is now an Associate Professor and the Postgraduate Director at the Department of Computer Science and Engineering in the Hong Kong University of Science and Technology. His research interests include wireless sensor network, peer-to-peer computing, and pervasive computing. Dr. Liu and his student Mo Li received the Grand Prize of Hong Kong ICT Best Innovation and Research Award 2007. He is a member of the ACM and a senior member of the IEEE.



Xiangyang Li received the M.S. (2000) and Ph.D. (2001) degree at Department of Computer Science from University of Illinois at Urbana-Champaign. He received his Bachelor degree at Department of Computer Science and Bachelor degree at Department of Business Management from Tsinghua University, P.R. China, both in 1995. He has been with Department of Computer Science at the Illinois Institute of Technology since 2000. Currently he is an Associate Professor of Department of Computer Science, IIT. His research interests span wireless ad hoc and sensor networks, non-cooperative computing, computational geometry, optical networks, and cryptography. He is an editor of Ad Hoc & Sensor Wireless Networks: An International Journal, and Editor of Networks: An International Journal. He has been a guest editor of special issues for ACM Mobile Networks and Applications, IEEE Journal on Selected Areas in Communications. He is a Member of the ACM and a senior member of the IEEE.



Panlong Yang received the B.S. degree, M.S. degree, and Ph.D. degree in communication and information system from Nanjing Institute of Communication Engineering, China, in 1999, 2002, and 2005 respectively. During November 2006 to March 2009, he was a postdoc fellow at the Department of Computer Science in Nanjing University. Currently he is an associate professor in the Nanjing Institute of Communication Engineering. His research interests include wireless mesh networks, wireless sensor networks and cognitive radio networks. He is a member of the IEEE Computer Society and the ACM SIGMOBILE Society.